(B) AMENDMENTS TO THE SPECIFICATION

1. Please replace paragraph [0020] with the following replacement paragraph (marked to show changes made):

[0020] The invention provides a platform for sampling, supervising and controlling the execution of multiple threads within a pipeline processor. The invention provides a powerful mechanism to direct and restrict the operation of multiple concurrent threads competing for more general system resources. In contrast to prior art embedded processor systems, the invention uses a pipelined architecture with a single processor/functional control unit wherein instructions take multiple processor cycles to execute and one instruction from an individual stream is typically executed each processor cycle. Unlike prior art systems, the invention provides a simple platform for sampling, supervising and controlling the execution of multiple <u>program</u> threads within a pipeline processor not through specialized hardware and memory registers but through any of the pipeline processor threads. This supervisory control function can also incorporate a hardware semaphore mechanism to control access to a set of program-defined resources including memory, registers and peripheral devices.

PAGE 3/27 * RCVD AT 10/16/2004 8:56:27 PM [Eastern Daylight Time] * SVR:USPTO-EFXRF-1/0 * DNIS:8729306 * CSID:7804242205 * DURATION (mm-ss):09-38

2. Please replace paragraph [0021] with the following replacement paragraph:

[0021] Multiple program threads are executed in parallel using a pipelined architecture and shared processor logic. By using a pipelined architecture the stages of fetching, decoding, processing, memory and peripheral accesses and storing machine instructions are separated and parallel program threads are introduced in a staggered fashion into the pipeline. At any time during pipeline execution, each separate thread machine instruction is at a different stage in the pipeline so that within any cycle of the processor, logical operations for "n" such threads are processed concurrently. Although eight clock cycles can process a single machine instruction (sixteen for ~~two word~~ two-word instructions) in a preferred embodiment of the invention, the efficiencies of the invention provide additional processing with eight threads. ~~On average~~ For one-word instructions, one complete machine instruction is completed per clock cycle from one of the active threads. The invention provides significant processing gain and supervisory functions using less than 100,000 transistors instead of the tens of millions of transistors found in non-embedded microprocessors.

3. Please replace paragraph [0022] with the following replacement paragraph:

[0022] Referring to Figure 1, single-chip embedded processor 10 has input/output capabilities comprising a central ~~eight thread~~ eight-thread processor core 12, clock input 14 with buffered output 16, various internal memory components shown as main RAM 18, a supervisory control unit (SCU) 20, peripheral adaptor 22 peripheral interface devices 24, an external memory interface 26 and a test port 28. The system is used for various embedded input/output applications such as baseband processor unit ("BBU") 30 connected to a RF transceiver 32 for communications applications and as an embedded device controller.

PAGE 4/27 * RCVD AT 10/16/2004 8:56:27 PM [Eastern Daylight Time] * SVR:USPTO-EFXRF-1/0 * DNIS:8729306 * CSID:7804242205 * DURATION (mm-ss):09-38

4. Please replace paragraph [0025] with the following replacement paragraph:

[0025] Supervisory control unit (SCU) 20 can be configured as a special purpose peripheral to work integrally with processor core 12 through peripheral adaptor 22. A "controlling" thread in processor core 12 issues input/output instructions to access supervisory control unit 20 by peripheral adaptor 22. Any of the processor threads can function as the controlling thread. Supervisory control unit 20 accesses various elements of processor core 12 as supervisory control unit 20 performs supervisory control functions. Supervisory control unit 20 supports various supervisory control functions including: 1) a run/stop control for each ~~thread~~ processor thread, 2) read/write access to the private state of each ~~thread~~ processor thread, 3) detection of unusual conditions such as I/O lock ups, tight loops, 4) semaphore-based management of critical resources, and 5) a sixteen-bit timer facility, referenced to master clock 14 for timing processor events or sequences. During normal processing supervisory control unit 20 reads state information from the processor pipeline without impacting program thread processing. Supervisory control unit 20 will only interrupt or redirect the execution of a program for a given thread when directed to by a controlling thread.

PAGE 5/27 * RCVD AT 10/16/2004 8:56:27 PM [Eastern Daylight Time] * SVR:USPTO-EFXRF-1/0 * DNIS:8729306 * CSID:7804242205 * DURATION (mm-ss):09-38

5. Please replace paragraph [0029] with the following replacement paragraph:

[0029] Processor core 12, shown in Figure 2, employs synchronous pipelining techniques known in the art to efficiently process multiple program threads concurrently. In one embodiment of the invention as illustrated, a typical single sixteen-bit instruction is executed in an eight-stage process. Where instructions consist of two sixteen-bit words, two passes through the pipeline stage are typically required. The eight stages of the pipeline include:

Stage 0	Instruction Fetch
Stage 1	Instruction Decode
Stage 2	Register Reads
Stage 3	Address Modes
Stage 4	ALU Operation
Stage 5	Memory or I/O Cycle
Stage 6	Branch/Wait
Stage 7	Register Write

PAGE 6/27 * RCVD AT 10/16/2004 8:56:27 PM [Eastern Daylight Time] * SVR:USPTO-EFXRF-1/0 * DNIS:8729306 * CSID:7804242205 * DURATION (mm-ss):09-38

6. Please replace paragraph [0030] with the following replacement paragraph:

[0030] There are several significant advantages to this pipelining approach. First, instruction processing is broken into simple, energy-efficient steps. Second, pipelined processing stages can be shared by multiple <u>processor</u> threads. Each <u>processor</u> thread is executing in parallel but at different stages in the pipeline process as shown in Figure 3. The vertical axis 50 in Figure 3 denotes the pipeline stage and the horizontal axis 52 corresponds to the processor clock 14 cycles or time. Although each instruction per <u>program</u> thread takes eight clock cycles to execute, on average the pipeline completes one instruction per clock cycle from one of the executing eight threads. Accordingly, the pipelined architecture provides significant processing gain. Third, since each of the ~~pipeline~~ <u>pipelined program</u> threads can be executed independently, real-time critical tasks can be dedicated to separate <u>program</u> threads to ensure their reliable execution. This feature of the invention is much simpler and more reliable than traditional interrupt-driven microprocessors where complex division of clock cycles between competing tasks is difficult to prove and implement reliably.

7. Please replace paragraph [0031] with the following replacement paragraph:

[0031] On each cycle of processor clock 14 the active instruction advances to the next stage. Following Stage 7, the next instruction in sequence begins with Stage 0. As seen in Figure 3, <u>program</u> thread 0 (T0) enters the pipeline Stage 0 in cycle "1" as shown by 54. As time progresses through the clock cycles, <u>program thread</u> T0 moves through Stages 0 to Stages 7 of the pipeline. Similarly, other <u>program</u> threads T1 to T7 enter the pipeline Stage 0 in subsequent cycles "1" to cycles "8" and move through Stages 0 to Stages 7 as shown in Figure 3 as <u>program thread</u> T0 vacates a particular Stage. The result of this hardware sharing regime is equivalent to eight thread processors operating concurrently.

PAGE 7/27 * RCVD AT 10/16/2004 8:56:27 PM [Eastern Daylight Time] * SVR:USPTO-EFXRF-1/0 * DNIS:8729306 * CSID:7804242205 * DURATION (mm-ss):09-38

8. Please replace paragraph [0033] with the following replacement paragraph:

[0033] The private state of each ~~thread~~ processor thread of processor core 12, as stored in the pipeline registers #0 to #7 (80 to 94) or the three-port RAM 36 module (registers 0 to 7, R0:R7), consists of the following: 1) a ~~sixteen-bit~~ sixteen-bit program counter (PC) register; 2) a ~~four-bit~~ four-bit condition code (CC) register, with bits named n, z, v, and c; 3) a set of eight ~~sixteen-bit~~ sixteen-bit general purpose registers (R0:R7); and 4) flags, buffers and temporary registers as required at each pipeline stage. Physically the general purpose registers can be implemented as a ~~sixty-four-word~~ sixty-four-word block in three-port RAM module 36 as seen in Figure 1. Register addresses are formed by the concatenation of the ~~three-bit~~ three-bit thread number (T0:T7) derived from the thread counter register, together with a ~~three-bit~~ three-bit register specifier (R0:R7) from the instruction word. A single ~~sixteen-bit~~ sixteen-bit instruction can specify up to three register operands.

9. Please replace paragraph [0034] with the following replacement paragraph:

[0034] As an instruction progresses through the hardware pipeline shown in Figure 2, the private state of each ~~thread~~ processor thread is stored in a packet structure which flows through the processor pipeline, and where the registers (R0:R7) are stored in the three-port, ~~sixty-four-word~~ sixty-four-word register RAM 36 and the other private values are stored in the Pipeline Registers #0 to #7 (80 to 94). The thread packet structure is different for each pipeline stage, reflecting the differing requirements of the stages. The size of the thread packet varies from forty-five bits to one hundred and three bits.

PAGE 8/27 * RCVD AT 10/16/2004 8:56:27 PM [Eastern Daylight Time] * SVR:USPTO-EFXRF-1/0 * DNIS:8729306 * CSID:7804242205 * DURATION (mm-ss):09-38

10. Please replace paragraph [0036] with the following replacement paragraph:

[0036] Similarly all eight <u>processor</u> threads have shared access to main RAM 18 and to the full peripheral set. Generally speaking <u>processor</u> threads communicate with one another through main RAM 18, although a given <u>processor</u> thread can determine the state of and change the state of another <u>processor</u> thread using supervisory control unit 20. In Stage 0 (62) and Stage 5 (72) the two-port main RAM 18 is accessed by two different threads executing programs in different areas in main RAM 18 as shown by 58 in Figure 4.

11. Please replace paragraph [0037] with the following replacement paragraph:

[0037] Referring to Figure 2, which illustrates the pipeline mechanism, the various pipeline stages and supervisory control unit 20 and thread counter 107 inter-working with core processor 12 pipeline is shown. The thread counter 107 directs the loading of a particular <u>processor</u> thread's state information into Stage 0 (62) of the pipeline and counts from 0 to 7 continuously. An instruction for a particular <u>processor</u> thread, as directed by the thread counter 107, enters the pipeline through Pipeline Register #0 (80) at the beginning of Stage 0 (62). The Instruction Fetch Logic 96 accesses main RAM 18 address bus and the resultant instruction data is stored in Pipeline Register #1 (82). In Stage 1 (64) the instruction is decoded. In Stage 2 (66) this information is used to retrieve data from the registers associated with the given thread currently active in this stage. In Stage 3 (68) Address Mode Logic 100 determines the addressing type and performs addressing unifications (collecting addressing fields for immediate, base displacement, register indirect and absolute addressing formats for various machine instruction types). In Stage 4 (70), containing the ALU 102 and associated logic, the ALU 102 performs operations (for address or arithmetic adds), sets early condition codes, and prepares for memory and peripheral I/O operations of Stage 5 (72).

PAGE 9/27 * RCVD AT 10/18/2004 8:56:27 PM [Eastern Daylight Time] * SVR:USPTO-EFXRF-1/0 * DNIS:8729306 * CSID:7804242205 * DURATION (mm-ss):09-38

12. Please replace paragraph [0039] with the following replacement paragraph:

[0039] Figure 2 also shows supervisory control unit 20 used to monitor the state of the processor core threads, control access to system resources, and in certain circumstances to control the operation of threads. Supervisory control unit 20 can selectively read or write state information at various points in the pipeline hardware as illustrated in Figure 2. It is not a specialized control mechanism that is operated by separate control programs but is integrally and flexibly controlled by any of the threads of processor core 12. Supervisory control unit 20 is configured as a peripheral so it is accessible by any thread using standard input/output instructions through the peripheral adaptor logic 104 as indicated by the thick arrow 105 in Figure 2. The formats of these instructions "inp" and "outp" are described later. When a given thread wishes to direct a thread-specific supervisory control unit 20 operation, it must first write a pointer value to input/output address "4" (112) as is shown in Figure 5. The pointer 112 contains the thread being accessed by supervisory control unit 20 in bit locations "3" to "5" (114) as shown in Figure 6. If a register is accessed through ~~an~~ <u>a</u> supervisory control unit 20 operation, the value of the desired register is contained in bits "0" to "2" (116) of the pointer.

PAGE 10/27 * RCVD AT 10/16/2004 8:56:27 PM [Eastern Daylight Time] * SVR:USPTO-EFXRF-1/0 * DNIS:8729306 * CSID:7804242205 * DURATION (mm-ss):09-38

13. Please replace paragraph [0040] with the following replacement paragraph:

[0040] Various supervisory control unit 20 read and write operations are supported. Read accesses ("inp" instruction) have no ~~affect~~ effect on the state of the processor thread being read. As shown in Figure 5, register values (R0:R7), program counter values, condition code values, a breakpoint (tight loop in which a thread branches to itself) condition for a given thread, a wait state (thread waiting for a peripheral to respond) for a given thread, a semaphore vector value and a continuously running ~~sixteen bit~~ sixteen-bit counter can be read. The "breakpoint" register 124 detects if a thread is branching to itself continuously. The "wait" register 126 tells if a given processor thread is waiting for a peripheral, such as when a value is not immediately available. The "time" register 130 is used by a thread to calculate relative elapsed time for any purpose such as measuring the response time of a peripheral in terms of the number of system clock cycles. By convention a given target thread should be "stopped" before any write access ("outp" instruction) is performed on its state values. If a controlling thread desires to change a register, program counter or condition code for a given target thread, the controlling thread must first "stop" the target thread by writing a word to stop address "3" (132) as seen in Figure 5. Bit "0" to bit "7" of the stop vector correspond to the eight threads of processor core 12. By setting the bit corresponding to the target thread to one, this causes the target thread to complete its current instruction execution through the pipeline. The pipeline logic then does not load any further instructions for that thread until the target thread's bit in the stop vector is once again set to zero by the controlling thread, such as in a "run" operation. Once the target thread is stopped the controlling thread can then write to any register value (138), the program counter (136) or the condition codes (134) of the target thread by performing a write ("outp" instruction) to the appropriate supervisory control unit 20 input/output address location as shown in Figure 5. This feature is useful in reconfiguring processor core 12 to perform in various SIMD and MIMD configurations (described later) or in otherwise modifying the target threads execution flow.

PAGE 11/27 * RCVD AT 10/16/2004 8:56:27 PM [Eastern Daylight Time] * SVR:USPTO-EFXRF-1/0 * DNIS:8729306 * CSID:7804242205 * DURATION (mm-ss):09-38

14. Please replace paragraph [0044] with the following replacement paragraph:

[0044] The get thread number instruction "thrd" 146 as shown in Figure 8 and later described is an important feature of the invention. The "thrd" instruction 146 is used by a given program thread to determine its identity or assigned processor thread number. Figure 9 illustrates an example of how this instruction, in combination with supervisory control unit 12 control registers, can operate processor core 12 in SIMD and then in MIMD configurations or vice versa. Such a capability can be used at any point during operation of processor core 12 to flexibly configure all or part of the ~~thread processors~~ processor threads in either SIMD or MIMD configurations or modes. When processor core 12 starts up from the power up state, the eight program threads supported by processor core 12 do not know their assigned processor thread ~~number~~ numbers. In the example, shown in Figure 9, processor core 12 uses all eight processor threads in parallel to initialize 16k words of Main RAM 18 memory to zero. Thread zero then stops all other processor threads, ~~reinitialized~~ reinitializes the program counter and register zero of the other processor threads and restarts them so that they begin executing eight independent programs on independent sets of data such as in MIMD operation. At the top of the example constants 150, corresponding to supervisory control unit 20 registers, are initialized. In section "Initialize Threads" 152 at the beginning of concurrent multithread operation, each program thread is sequentially loaded into the pipeline starting from processor thread zero. The "thrd" instruction 146 is used by each program thread to determine its corresponding processor thread number. Register two (r2) for each processor thread is also set to zero. In the section "InitMemory" 154 each processor thread stores zero to the address located in their register zero (r0) and then increments the memory location address by eight words. This is preferred to coordinate eight threads concurrently writing to a sequential memory block of eight words. When 16k words of main RAM 18 memory are initialized, the program exits from "InitMemory" 154. In "StopThreads" 156, program thread zero writes to the supervisory control unit 12 stop register 132. This writes a zero to the bit corresponding to thread zero. Ones are written to the other threads. This causes the execution of all other threads except thread zero to halt once the current instruction in the pipeline is completed. In sections "InitForMIMD" 158 and "SetMIMD" 160, thread zero initializes the program counters of the stopped

PAGE 12/27 * RCVD AT 10/16/2004 8:56:27 PM [Eastern Daylight Time] * SVR:USPTO-EFXRF-1/0 * DNIS:8729306 * CSID:7804242205 * DURATION (mm-ss):09-38

threads, one to seven, to correspond to the beginning of the "MIMDStart" section 162 and initializes the register zero of each thread to contain a value corresponding to separate independent program segments at different memory addresses. The address of a given instruction is shown in the left-most column 151 of Figure 9. From this column 151 we see that the MIMDStart section 162 starts at address 23. At the end of the "SetMIMD" section 160, thread zero starts the stopped threads one to seven by changing the value of supervisory control unit 20 stop vector 132. All threads then begin executing at the beginning of section "MIMDStart" 162 and based upon their different register zero values, branch to independent program segments where they operate independently. The program segment thus shows how the "thrd" instruction 146 and supervisory control unit 20 stop/run function can be used to configure processor core 12 threads for MIMD and SIMD modes of operation. ~~Threads~~ Processor threads in processor core 12 may operate in a mixed mode where some threads are executing in a SIMD configuration and others are operating in a MIMD configuration.

15. Please replace paragraph [0047] with the following replacement paragraph:

[0047] The "c" condition code efficiently detects words having a value of ~~"0"~~ "1" to "255". One application of this feature is the detection of ~~one byte~~ one-byte data types such as characters. Such detection can be done without expending additional machine cycles for value testing.

PAGE 13/27 * RCVD AT 10/16/2004 8:56:27 PM [Eastern Daylight Time] * SVR:USPTO-EFXRF-1/0 * DNIS:8729306 * CSID:7804242205 * DURATION (mm-ss):09-38